Using Meta-patterns to Construct Patterns

Rébecca Deneckère

Université Paris 1 Panthéon-Sorbonne, Centre de Recherche en Informatique
90 rue de Tolbiac 75013 PARIS, France
tel : + 33 (0) 1 44 07 86 34, fax : + 33 (0) 1 44 07 89 54,
denecker@univ-paris1.fr

Abstract. The pattern notion defines techniques allowing the existing knowledge reuse. Usually, the knowledge encapsulated in these patterns is stored in classic library repositories that quickly become overcrowded. To solve this problem, [1] proposes the use of process maps in order to organize and select them. But the completeness of the maps is a very important problem that has to be solved in order to offer a useful guidance to the method engineer. This paper proposes a guideline pattern construction technique guiding engineers when creating the maps.

1 Introduction and State of Art

The pattern notion has been widely used these last years. As a result, the patterns repositories number increases and it is more and more difficult to manage them. A way to solve this problem is the process map technique utilization that helps the method engineer to organize and select patterns. This section described these concepts.

The concept of **pattern** is very present in the literature and in a lot of different domains [2][3][4][5][6][7][8]. In [9] a pattern is described as "*a problem which occurs over an over again in our environment and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing the same twice*". In Method Engineering, generic patterns aim at proposing a mean for constructing situation specific methods. Such patterns allow to know which are the best processes in a specific situation and they guide the method engineer in the construction of a specific method.

A pattern description must include the *problem* for which the pattern proposes a *solution* and the recurring set of situations in which the pattern applies [10]. We use here the formalism detailed in [11][12]. A pattern contains two parts: the reusable knowledge (the body) and the application aspects of this pattern (the signature). The pattern's *body* encapsulates the process description to apply on the product under modification. The signature is described to represent the situation before the modification, the intention to achieve and the

target of that modification. We also call this concept *interface* [12]. It is seen as a triplet <situation, intention, target> associated to a body.

Following the appearance of a lot of transformation patterns catalogues, we have pointed out in [1] the problems of storing these patterns without order. Firstly, the patterns are stored in a library but this one may rapidly become overcrowded as engineers add new patterns as time goes by, and, secondly, some patterns have precedence relationships and require a way to introduce and execute them in predefined order. [1] has proposed an organizational technique to solve these problems by using **process maps**.

These maps are a formalization of the utilization process of the patterns that help the engineer with a guidance of each transformation. [1] uses the technique of process map [14][15] in order to sort the patterns in a catalogue. It is a labeled graph composed of nodes and edges. Nodes represent intentions that the engineer wants to reach and edges manners for reaching these intentions. The directed nature of the graph shows possible intentions dependencies. An edge enters a node if its manner can be used to achieve its intention. Since there can be multiple edges entering a node, the map is able to represent all the manners that can be used for achieving an intention. An execution path, determined by a map, always starts by the intention *Start* and ends by the intention *Stop*.

In [1], we have defined that a pattern, contextually to a specific map, may be represented by a defined section. Each of these sections (i.e. each of these patterns) represents a specific way to reach a target intention, from a node in the map, by the execution of a particular manner. The next figure shows the equivalence between concepts of *Section* and *Pattern* using the representation of two sections of a map.

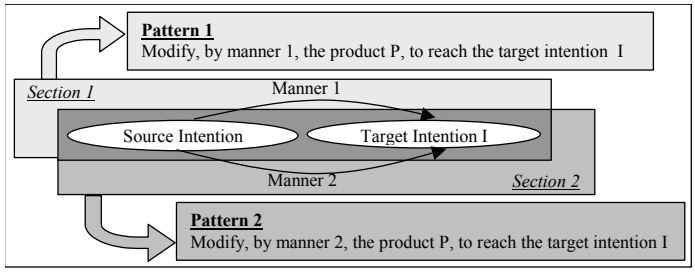

Figure 1: Sections and patterns

This figure illustrates the fact that the applicable pattern between two intentions will be different, according to method engineer's manner choice. Then, we may say that it exists, from a specific source intention to a specific target intention, as many applicable patterns than manners.

The problematic and proposed solution is described in the next section. An application of this technique to the OO method extension approach is done in the section three. We conclude in the fourth and last section.

2 Problematic and Proposed Solution

The major problem left was the map completeness. The process maps are perfects if they are complete, so the method engineer knows all the alternatives that are offered to him. However, if some patterns are not present on the map, engineers may come to a dead-end and all the usefulness of this technique is thrown away. In order to construct complete maps, we propose here a specific construction pattern technique using meta-patterns.

Problematic when constructing a map, for a specific application domain, begin with the explicit inventory of all the intentions. For instance, if we want to extend a method, each intention will represent a specific element integration into this specified method. Then, the method engineer makes the inventory of the applicable manners to reach these intentions. Finally, each of these sections has to be described by a pattern. However, manners inventory in a map is a very long, delicate and strategic work and represents the major difficulty of that process. If the method engineer forgets one section and that the extension process of a specific method need this one to reach its goal, this extension can't go farther. To solve that case, this paper proposes to create a set of generic manners specific to the approach used. Apply all generic manners to all map intentions allows map completeness certification.

Furthermore, in order to ease the sections realization of the map (i.e. the description of the associated patterns), all the relative knowledge of the patterns construction is encapsulated in a specific meta-pattern. The method engineer, as soon as the elements to integrate will have been identified, will directly realize the map intentions, by applying all meta-patterns defined. Each meta-pattern application will guide to a pattern construction allowing to reach a target intention, from a source intention, by applying a specific manner on it. This construction process allows to guide the method engineer when identifying the map sections and also when constructing the associated patterns. The next figure shows the instantiation of a meta-pattern using the *specialization* generic manner for the construction of three patterns (i.e. three sections of the map). A complete description of this meta-pattern is done in [11].

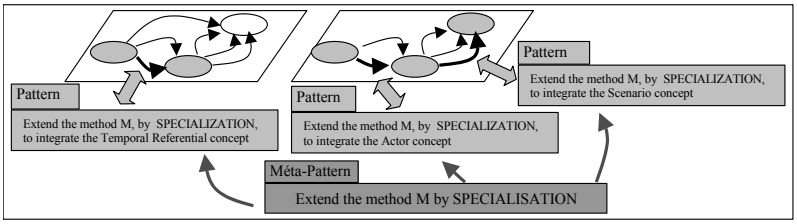

Figure 2: Meta-pattern instantiation to obtain patterns

To illustrate this paper, we choose to apply this technique to the method extension approach and we will also only focus on the set of patterns that may be used when extending an OO method.

3 Application to the OO Method Extension Approach

The method extension approach has been defined in [11]. It is described as a *technique allowing to take into account more things that was in the origin set*. The manners and meta-patterns used here are specific to this particular approach.

The manners studied in this paper allow modifying a method. Three parameters are considered: (a) the method element to modify - [Element X] (this element is a part of the method before the extension), (b) the element that the method engineer wishes to integrate into the method - [Element Y] (this element is a part of the method after the extension) and (c) the element likely to be of no use after the extension – [Element D] : *Manner ([Element X], [Element Y], [Element D])*. Note that the two first arguments are mandatory. The third one – [Element D] – is optional. All extensions don't change the source method to the point of leaving an useless element. In that last case, it is necessary to leave the possibility for deleting this element and guaranty the method coherency.

Lets take the example of a pattern that allows integrating the *Calendar Class* concept into a method that already contains the *Clock Class* concept. In fact, it will be implemented as a replacement of the first one by the second one. The situation before the extension is : *Clock Class = Inherits of (Class), Composition (Temporal Event)*[1]. The manner application is : *REPLACEMENT (Clock Class, Calendar Class, ϕ)*. The situation after the extension is : *Calendar Class = Inherits of (Class), Composition (Temporal Event)*.

The knowledge relative to each manner is encapsulated into a specific pattern named "*Meta-pattern*". As exposed before, the meta-patterns follow the patterns description. They are composed by a signature (Figure 3) and a body. As for the patterns, the meta-patterns may be differentiated following two specific types: the ones allowing to construct a Product pattern and the ones allowing to build a Process pattern.

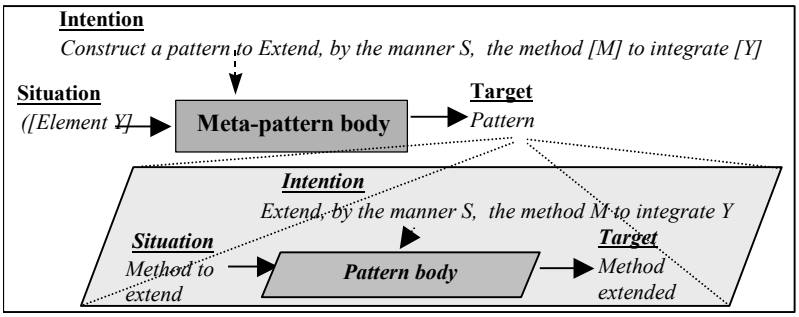

Figure 3: Meta-pattern interface

[1] This formalism is described in [11].

The meta-pattern situation represents the Element Y (the additional element we want to integrate into the original method). The meta-pattern intention is to build a pattern that will integrate that element in the method following a specific manner, resulting a target meta-pattern that is a pattern for a specific element and following a specific manner.

The meta-pattern body represents all the operations allowing the construction of the pattern, i.e. the definition of its situation, its intention, its target and its body. This body is constructed by a set of operators corresponding to the manner used. Each manner represents a set of modification operators containing three unknown values : Element X, Element Y and Element D (the three arguments of the manner). Element Y is known when we use the meta-pattern in order to build a pattern. However, Element X and Element D will only be known when instantiating this pattern to a specific method.

Lets illustrate that concept with a sample. The following figure shows the successive instantiations, from a meta-pattern to a pattern (for an integration of a specific element) and to a pattern allowing the extension of a specific method. *Calendar Class* is the concept to integrate in the O* method.

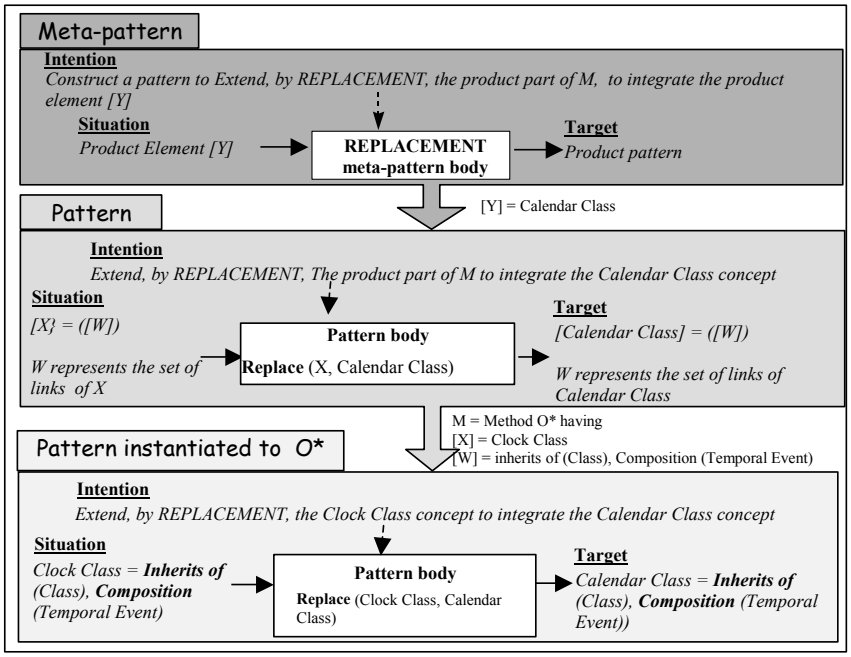

Figure 4: Example of pattern construction with meta-patterns

This figure shows the two steps allowing to construct a pattern, given a specific method. The first step represents the meta-pattern instantiation for the *Calendar Class* concept. The second step represents the pattern instantiation to the O* method.

As the objective in this paper is automating patterns building, we will focus on the inventory of all the possible manners to extend a method (the manners set), i.e. all the possible meta-patterns. We will describe here differences between the two types of meta-patterns (Product and Process) and theirs related manners. To limit the research domain of this work, we will only focus on the set of patterns that may be used when extending an OO method. These manners will be called OOME manners (Object Oriented Methods Extension manners). On the same way, the associated patterns are called OOME patterns.

3.1 OOME Product Manners

In case where Product part of a method doesn't match method engineer requirements, that method may be modified by applying a Product pattern defined in the patterns catalogue. This catalogue is populated with the Product meta-patterns application. These specific patterns allow constructing pertinent patterns adapted to the application at hand. They have to be applied on the method in order to modify it. The modified method will then handle a bigger set of concepts to construct a more complete and coherent application.

A Product meta-pattern may be described with its interface. The situation is the Product element to integrate in the method. The intention is the construction of a pattern to extend the method. The target is that pattern, created for a specific product element and following a selected manner. The body is the set of creation operations allowing to create the pattern. The body of the pattern created will be composed of a set of transformation operations (corresponding to the manner chosen).

As a method extension is a *technique allowing to take into account more things that was in the origin set*, a Product part extension will introduce new concepts into an existing model. That merge may be made following two different ways : either the concept to integrate is a new element, or it is a concept already represented in the method but in an incomplete or incorrect way.

Introduction of a New Concept in the Method

The introduction of these new concepts had to be followed by their connection to the rest of the model. A concept is important only if it is linked with other concepts. These grafts are realized by instantiating the different kind of links defined in the Product meta-model of the method. Note that the extension process will be different following these links. As a result, the extensions may be defined according to the link type used to connect the new element to the method model. A generic links set has been defined with the study of several OO methods (OMT[16],OOA&D[17], O*[18], etc) and results in the fact that a new concept may be added in a method product model by connecting it to an other concept with the inheritance, composition or association link.

- *Inheritance Link:* The method engineer may integrate a new concept in the model by connecting it to an existing concept with an inheritance link. The new concept is then inserted as a specialization of an existing or a non-existing concept.

In the first case, the new concept is inserted and the inheritance link between it and the existing concept is added. This technique may be used only if the method already contains a concept that may be viewed as a super-type of the concept to integrate. Lets use an example concerning an extension of the OMT method. Its Product model contains the *Event* concept. The insertion of the *Temporal Event* new concept may be made by a specialization of *Event*. Integrating the *Temporal Event* concept, and connecting it by specialization with the *Event* one, makes the extension.

On the contrary, in the second case, the method does not contain any concept of this kind. However, it contains one with a semantic similar to the concept to integrate. In this case, the extension allows integration of an other element that de facto generalize existing and new concepts. Lets consider again the OMT method. It contains the *Object Class* concept. The insertion of the *Actor Class* may be made with that technique, regarding their semantic similarity. Firstly, the extension of the method will insert a generalization of the *Object Class* that we call the *Class* concept. Next, the *Actor Class* concept is integrated. Finally, this new concept is connected by a specialization link to the *Class* concept.

To insert a concept by an inheritance link may be viewed as two different techniques : inheritance link usage or generalization link usage. Each of these two techniques represents a specific OOME Product manner : SPECIALIZATION and GENERALIZATION.

- *Composition Link:* The method engineer may also integrate a new concept in the model using a composition link. The new designed concept may be inserted as a composed concept or as a component of an existing concept.

In the first case, the concept is added in the target Product model then connected as a composed element of an existing concept. Lets illustrate that with an OMT method case. It contains the designed *Object Class* concept, composed of *Property*, *Event* and *Operation* concepts. The method engineer may extend this method by integrating the *Constraint* concept on the *Object Class*. The best way to integrate this concept is to consider it as a component of the *Object Class*, just as the three others.

In the second case, on the contrary, the new concept doesn't become a *component* of an existing concept but a *composed* concept of an existing one. For instance, a method that contains the concept of *External Event* and where the method engineer wants to integrate the *Actor Class* concept. Then, process first step is the concept *Actor Class* integration in the model. Next, the concept of *External Event* is connected to this new concept using a composition link.

To integrate a concept with this link represents a composition link usage or a decomposition link usage. Each of these two different techniques represents a specific OOME Product manner : COMPOSITION and DECOMPOSITION.

- *Association Link*: An other existing link in the OO methods generic meta-model is association. The method engineer may insert a new element and connect it to the model with this kind of link. In that case, the element is inserted and is connected with an existing concept. Lets take the example of a method containing the *Action* concept. The method engineer wants to integrate the *Agent* concept into that one. This problem may be solved by using an association link (An *Action* is done by an *Agent* and an *Agent* makes one or more *Action*(s)).

This kind of insertion represents the fact that the new concept is related to an existing concept. This is the last OOME product manner : ASSOCIATION.

Introduction of a Concept Incompletely Represented in the Method

It is also possible to extend a model by replacing an existing concept in order to change its description. Lets use the example of a method that contains the *Granule* concept. The method engineer wants to improve this notion of *Granule* by adding the *Calendar* concept. The existing *Granule* concept becomes incomplete according to the method engineer requirements. A way to solve this problem is to replace the existing concept by the new one.

To extend a method with this particular technique is characterized by the application of a concept replacement. This way of working represents a specific OOME product manner : REPLACEMENT.

To resume, we may enumerate the following set of OOME Product manners : Inheritance, Generalization, Composition, Decomposition, Association and Replacement. Each of them will be supported by a Product meta-pattern.

3.2 OOME Process Manners

If the method engineer applies a Product pattern and that the extended method contains a Process part, it is better to also extend this part of the method. It will improve the method completeness. On the same way as we defined Product meta-patterns, it is possible to define some Process meta-patterns.

Like Product meta-patterns, Process meta-patterns may be described with their interface. The situation is the construction concept to integrate in the method. The intention is the construction of a Process pattern in order to extend that method. The target is this Process pattern, created for a specific product element construction, following a certain manner. The body is the set of creation operations allowing to create the pattern (situation, intention, target, body). The created pattern's body will be composed of a set of transformation

operations (corresponding to the OOME manner chosen). This section describes OOME process manners.

Extension of a method's Process part is made by grafting new processes in the process tree[2] of the existing method, in order to take into account the new concepts construction. There is two possibilities supplied to the method engineer: either the schema construction step is left untouched and grafts are made on a very controlled way, or the process tree is modified in order to automatically integrate all new processes.

Controlled Integration

Processes allowing the method extensions are grafted in order to leave the existing construction process unchanged. The method engineer will extend the schema elements only when all of them will have been fully defined first. The construction process tree keeps its construction processes (« Construct X ») but is incremented of a new extension process (« Extend X »). These two processes are executed in sequential order. We call that technique a Sequential Extension.

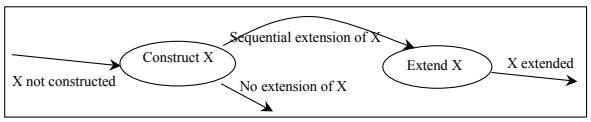

Figure 5: Global and local sequential extension principles

The principle shown on the Figure 5 represents the sequence between the construction process and the extension process of X. However, the *X* term may have a different signification according to the needed granularity. At the lower level, X represents a simple element of the method to extend, i.e. the construction of the *Class*, *Granule* or others concepts. On the contrary, at the highest level, X represents the entire construction schema of that method.

In the first case, the process tree puts into sequence the entire construction of the schema with its extension. In the second case, an extension represents a sequence in the process tree that allows step forward from the construction of a simple element to its extension. As shown in the previous figure, these two possibilities represent two specific Process manners : LOCAL SEQUENTIAL EXTENSION and GLOBAL SEQUENTIAL EXTENSION.

Transparent Integration

This type of extension doesn't leave the construction process unchanged. On the other hand, the new processes are fully integrated in the existing tree

[2] The formalism used to represent the Process part of the methods is the one of the Esprit project « NATURE » described in [19][20][21]

process. Extension doesn't appear as an artificial step but as a more complete step. In consequence, the method engineer doesn't work in a sequence <construction, extension> but simply executes a process tree taking the modifications into account from the beginning. This principle is illustrated in the Figure 6.

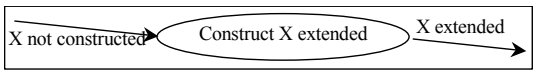

Figure 6: Integrated extension principle

That kind of extension represents the specific Process manner INTEGRATED EXTENSION.

A Process part method may be extended by one of these three following process OOME manners : Local sequential extension, Global sequential extension or Integrated extension. Each of these manners will be represented as a Process meta-pattern.

Process OOME Manners *Versus* Product OOME Manners

We have to note that an OOME Product manner application will have effects on the OOME Process manners application and induce three possible impacts from it. (1) With a *specialization* or a *generalization*, there will be a modification of the process tree alternatives concerned by the element. It will have an augmentation of the specialized types or an additive step. (2) On the same way, with a *composition* or a *decomposition*, it will guide to a modification of the constructing sequence of a Product element. (3) A *replacement* will leads to a graft of a new branch replacing an old one.

We may also differentiate two cases for the *global sequential extension*. Either the desired extension is the first one to be grafted or it is not the case. If it is the first extension to be realized, the method engineer has to completely reevaluate the precedence graph of the extension sequence in order to guaranty method's coherence and integrity. We call these two types: "First graft" and "Additional graft".

Notice that extensions based on *specialization* and *generalization* will be different according to the specific specialization of the considered element. Then, we may consider two types of specialization: "by type" or "by state". Al thought a type specialization leads to a complete differentiation of all the specialization of a concept, a state specialization will lead to the same description, except the type of the concept that will be different.

We defined several Process manners that can be used after the defined Product manners. However, it is not always possible to use them whichever Product manner has been previously used. Moreover, according to the Product manner, the Process modifications will be different. The Figure 7 shows the different possibilities of combinations.

Process EM manner / Product EM manner	SEQUENTIAL EXTENSION			INTEGRATED EXTENSION	
	GLOBAL		LOCAL	Element having a type specialization	Element having a state specialization
	First graft	Additional graft			
SPECIALIZATION	GLOBAL SPECIALIZATION (First graft)	GLOBAL SPECIALIZATION (Additional graft)	LOCAL SPECIALIZATION	INTEGRATED SPECIALIZATION (by type)	INTEGRATED SPECIALIZATION (by state)
GENERALIZATION	GLOBAL GENERALIZATION (First graft)	GLOBAL GENERALIZATION (Additional graft)	LOCAL GENERALIZATION	INTEGRATED GENERALIZATION (by type)	INTEGRATED GENERALIZATION (by state)
COMPOSITION	GLOBAL COMPOSITION (First graft)	GLOBAL COMPOSITION (Additional graft)	LOCAL COMPOSITION	INTEGRATED COMPOSITION	
DECOMPOSITION	GLOBAL DECOMPOSITION (First graft)	GLOBAL DECOMPOSITION (Additional graft)	LOCAL DECOMPOSITION	INTEGRATED DECOMPOSITION	
ASSOCIATION	GLOBAL ASSOCIATION (First graft)	GLOBAL ASSOCIATION (Additional graft)	LOCAL ASSOCIATION	INTEGRATED ASSOCIATION	
REPLACEMENT				INTEGRATED REPLACEMENT	

Figure 7: Process OOME manners *Versus* Product OOEM manners

Notice that we didn't allowed to use a *sequential extension* if the OOME Product manner used was a *replacement*. It will not be very useful to construct an element if we have to replace it as soon as we extend the schema.

As a result, we obtain 23 specific OOME Process manners that we may use on an OO method.

4 Conclusion and Future Works

We proposed here a technique to construct patterns with methodical guidelines, guarantying completeness relative to map construction. As a matter of fact, if the process maps aren't complete, it may lead to coherency problem of the modified method. To solve this problem, we defined a set of manners that are encapsulated in specific patterns called meta-patterns. These meta-patterns lead to the construction of all patterns of process maps. As a result, all sections are defined, the method engineer can't reach a dead-end and the map is complete.

We have illustrated this technique with the Extension approach applied on Object Oriented methods. This specific application field leads us to define five Product meta-patterns and twenty-three Process meta-patterns that may be used to integrate new elements (concept and construction) into OO methods.

Despite that technique, following difficulties remain to be solved:

- A larger application field: The application of this technique to other domains than the OO method extension.
- The guidance supported by tools: a first one will help the construction of the map (and its related patterns) with the technique described in this paper and a second one will help its execution.

References

1. Deneckere, R., Souveyet, C. : Organising and Selecting Patterns in Pattern Languages with Process Maps. Proceedings of OOIS'2001 Conference. Springer-Verlag, Calgary (Canada) (2001)
2. Coad, P.: Object-Oriented Patterns. Communications of the ACM, Vol. 35, No. 9 (1992). pp 152-159
3. Beck, K.: Smalltalk, Best Practice Patterns. Volume 1, Coding. Prentice Hall, Englewood Cliffs, NJ. (1997)
4. Buschmann, F., Meunier, R., Rohnert, et al.: Pattern-Oriented Software Architecture - A System of Patterns. John Wiley (1996)
5. Coplien, J.O., and Schmidt, D.O. (ed.) : Pattern Languages of Program Design. Addison-Wesley, Readind, MA. (1995)
6. Gamma, E., Helm, R., Johnson, R., et al.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley (1994)
7. Hay, D.: Data Model Patterns: Conventions of Thought. Dorset House, NY (1996)
8. Fowler, M.: Analysis Patterns: Reusable Object Models. Addison-Wesley, (1997)
9. Alexander, C., Ishikawa, S., Silverstein, M., et al.: A Pattern Language. Oxford University Press, New York (1977)
10. Software Patterns. Communications of the ACM, Volume 39, No 10, (October 1996)
11. Deneckere, R. : Approche d'extension de méthodes fondée sur l'utilisation de composants génériques. PhD thesis, University of Paris1-Sorbonne (2001)
12. Deneckere, R., Souveyet, C. : Patterns for extending an OO model with temporal features. Proceedings of OOIS'98 conference. Springer-Verlag, Paris (France) (1998)
13. Rolland, C., Plihon, V., Ralyté, J.: Specifying the reuse context of scenario method chunks. Proceedings of the conference CAISE'98, Springer-Verlag, Pisa Italy (1998)
14. Benjamen, A.: Une approche multi-démarches pour la modélisation des démarches méthodologiques. PhD thesis, University of Paris 1, Paris (1999)
15. Rolland, C., Prakash, N., Benjamen, A.: A multi-model view of process modelling. Requirements Engineering Journal, p. 169-187 (1999)
16. Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W. : Object-Oriented Modeling and Design. P.-H. I. Editions, Eds. (1991)
17. Martin J., Odell J.: Object-Oriented Analysis and Desig.n P.-H. I. Editions, Eds. (1992)
18. Brunet J. : Analyse Conceptuelle orientée-objet. PhD Thesis, University of Paris 6. (1993)
19. Rolland, C., Souveyet, C., Moreno, M.: An Approach for Defining Ways-Of-Working. Information Systems, Vol 20, No4, pp337-359 (1995)
20. Rolland, C., Plihon, V.: Using generic chunks to generate process models fragments. Proceedings of the 2nd IEEE International Conference on Requirements Engineering, ICRE, ICRE'96, Colorado Spring (1996)
21. Jarke, M., Rolland, C., Sutcliffe, A., Dömges, R. (Hsrg.): The NATURE of Requirements Engineering. Shaker Verlag, Aachen (1999)